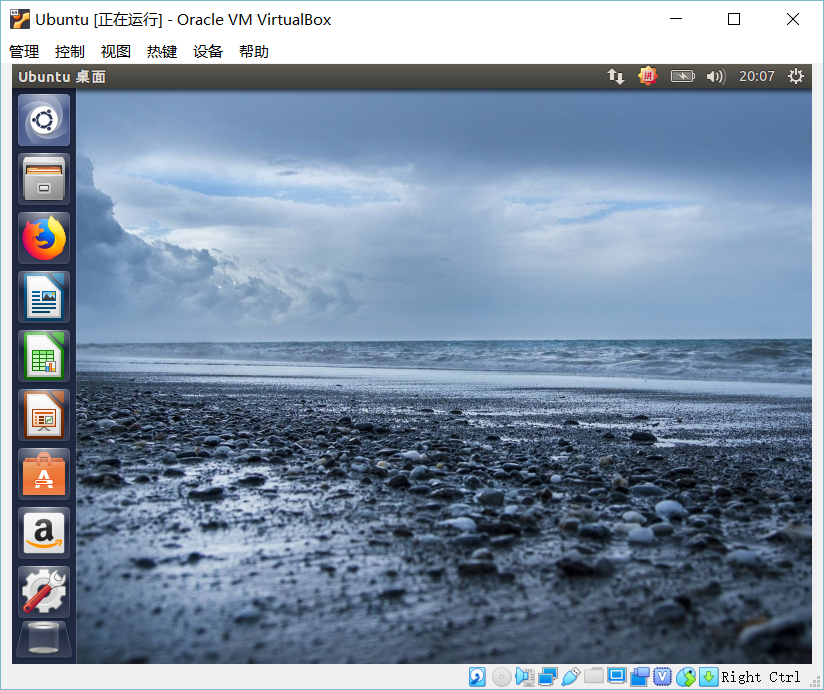
# 安装和配置Linux虚拟机

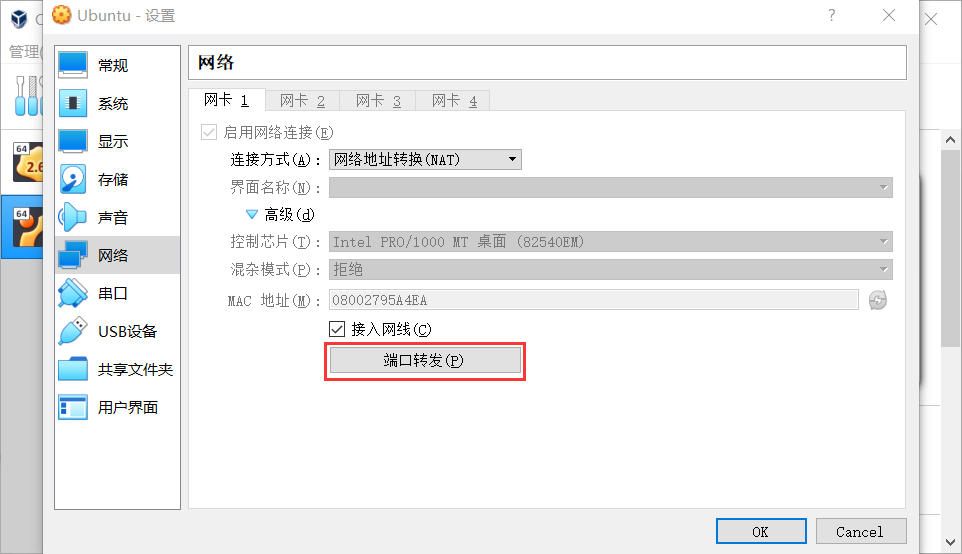
虚拟机安装比较简单，推荐使用Ubuntu镜像文件（[下载地址](http://mirrors.163.com/)），界面比较Friendly。

安装完成后的样子：

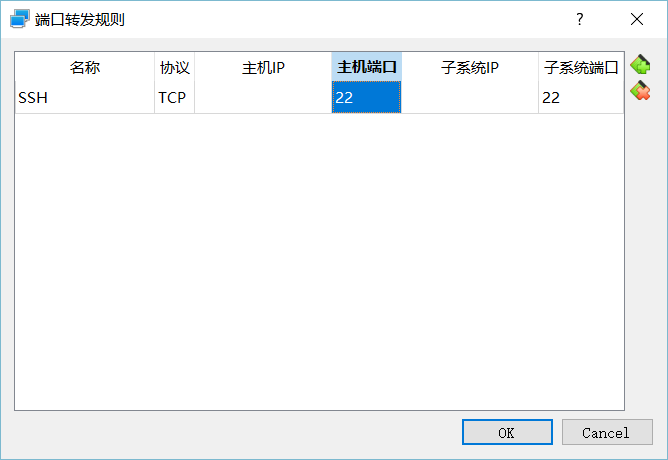


# 设置本地和虚拟机的SSH连接

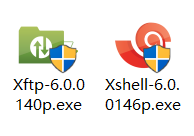
## 2.1设置虚拟机的网络：



端口就默认为22，不用改了



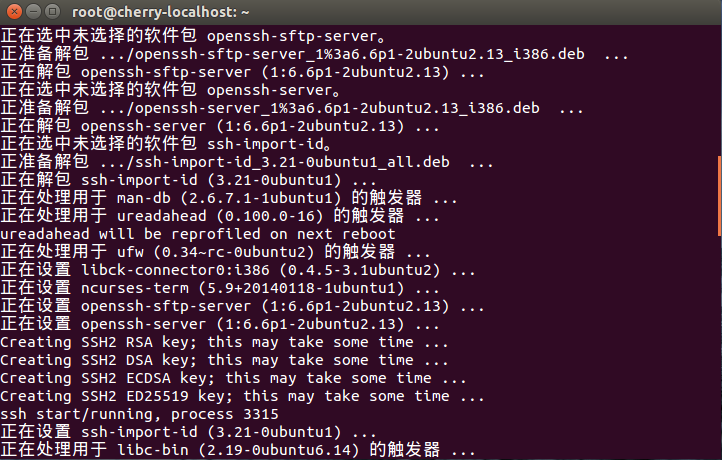
然后启动XShell，一个终端模拟软件，功能非常强大，和Xftp配套使用。

，XShell用来连接和交互，Xftp用来传输文件。

接着虚拟机中打开终端（快捷键为Ctrl+Alt+T），输入“sudo apt-get update“

然后输入当前账户的用户名的密码（就是让你用root账户登录）.





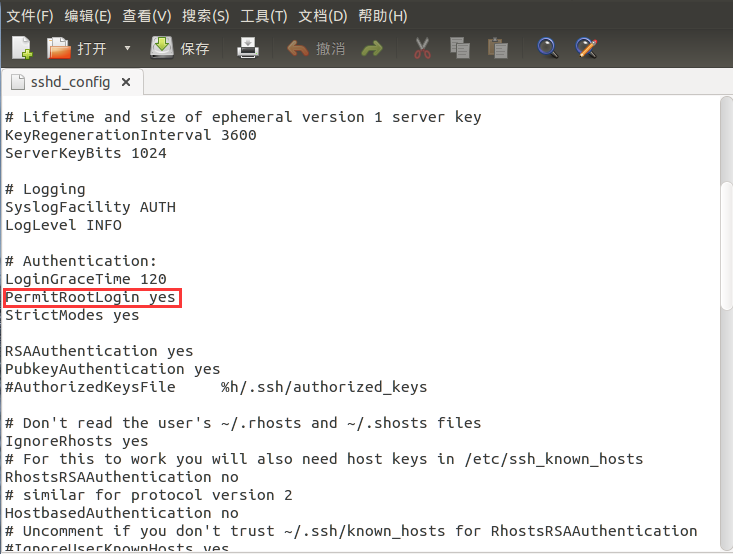
## 2.2安装和设置SSH服务

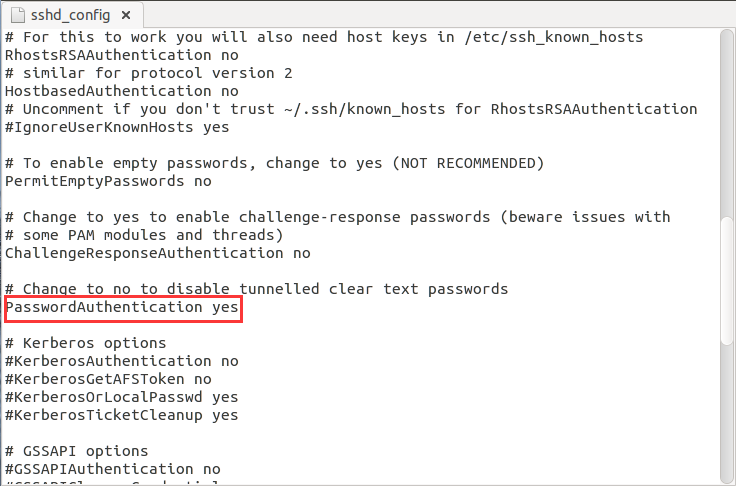
在 "终端窗口" 输入 "sudo apt-get install openssh-server"，然后输入y，安装完成。

在 "终端窗口" 输入 "service ssh status”观察ssh是否启动。如下状态说明已经启动：或者：打开"终端窗口"，输入 "sudo ps -e | grep ssh" --> 回车 --> 有 sshd，说明 ssh 服务已经启动，如果没有启动，输入 "sudo service ssh start" --> 回车 --> ssh 服务就会启动

## 2.3修改SSH设置

打开"终端窗口"，输入"sudo gedit /etc/ssh/sshd\_config"-->回车-->把配置文件中的找到#PasswordAuthentication no的注释#去掉，并且将NO修改为YES->找到#PermitRootLogin prohibit-password修改为 PermitRootLogin yes保存，修改成功。即修改为如下状态：

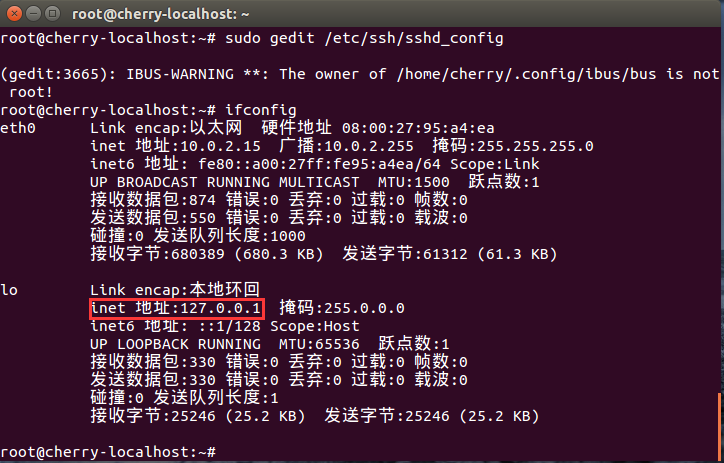


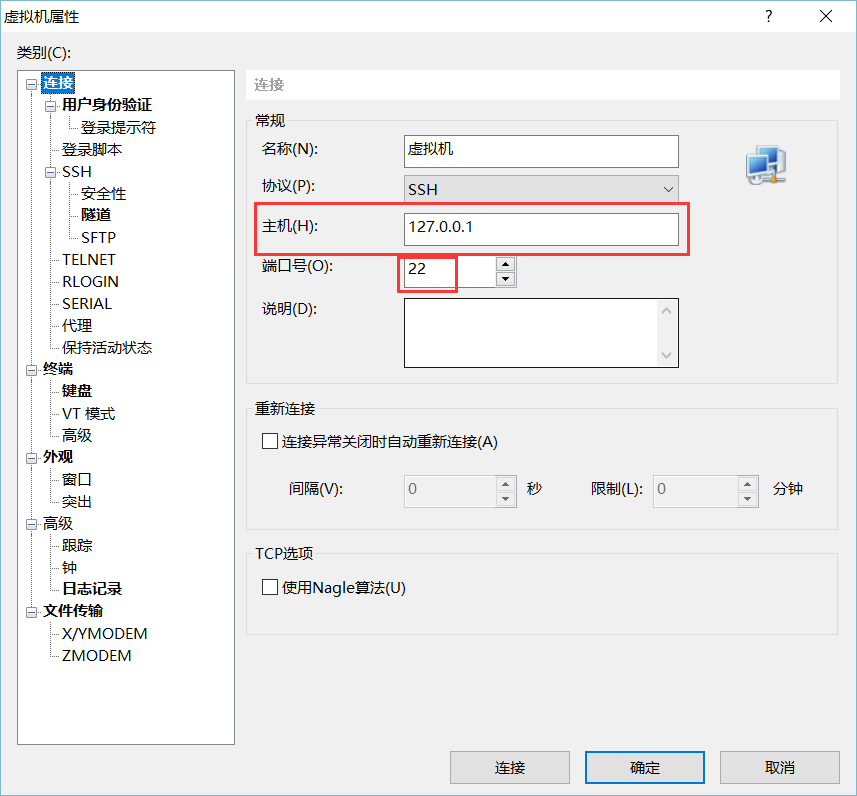


保存后退出，然后输入 service ssh restart 重启服务，这样就设置成功了。

## 2.4 设置SSH 连接

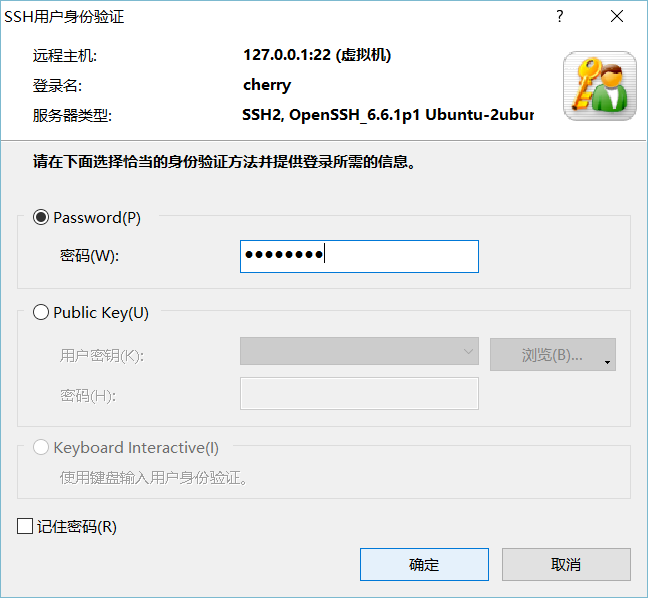
打开XShell，新建会话，在虚拟机中输入ifconfig查看IP，然后在主机名中输入IP和端口号：

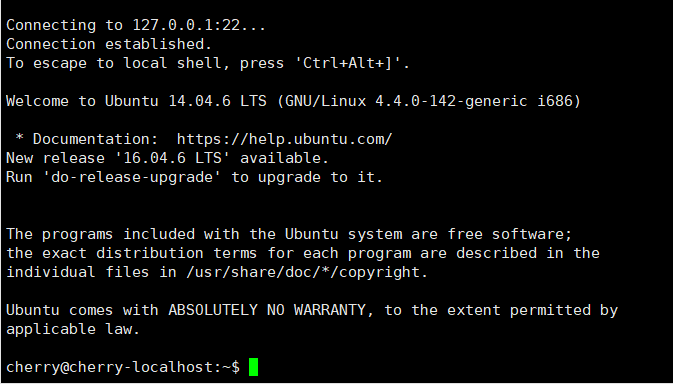




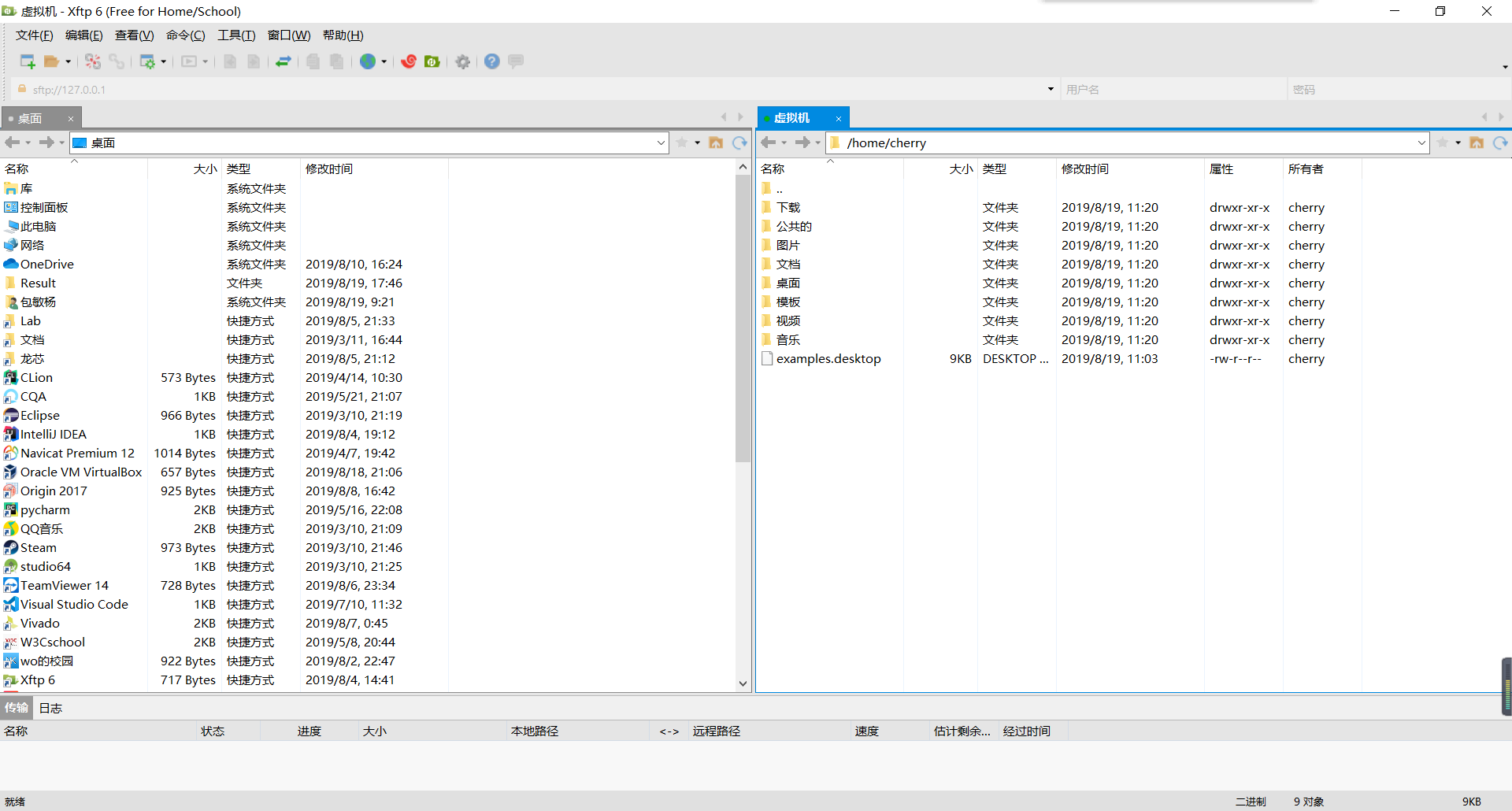
点击确定，输入虚拟机的用户名和密码，连接成功。







同理，连接完成的Xftp如图所示：



如果发生传送失败，记得在虚拟机中修改文件夹权限

在终端键入：sudo chmod 777 /opt

其中chmod 777表示文件的r,w,x（读、写、执行权限），如果给予全部三个权限，这三个值便全是1，那么111用十进制表示就是7。

设置完之后应该就可以正常传送文件了。

# GNU工具链的安装

使用的是龙芯公司的gcc-4.3-ls232.tar.gz，

[下载地址](http://ftp.loongnix.org/toolchain/gcc/release/gcc-4.3-ls232.tar.gz)：（<http://ftp.loongnix.org/toolchain/gcc/release/gcc-4.3-ls232.tar.gz>）

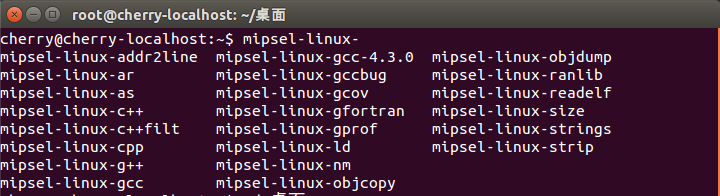
### 我们开始配置gcc

进入模拟器/opt文件夹，在终端输入tar -zxvf gcc-4.3-ls232.tar.gz，

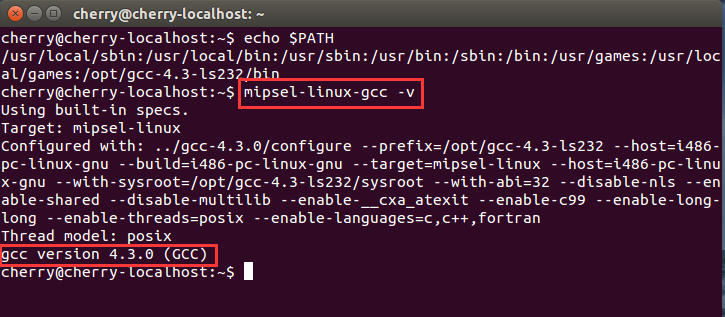
压缩完成。在home目录下查看隐藏文件 .bashrc ，在文件最后一行加上

export PATH=”$PATH:password/opt/gcc-4.3-ls232/bin”

再次在终端输入：mipsel-linux-，并按两次TAB，如图，表示系统路径设置正确



再次键入：mipsel-linux-gcc -v，如图：表示GCC工具链安装完成



至此，一些基本的准备工作就完成了，但是在此过程中会因为不熟悉Linux系统而导致一系列问题，在此总结一下。

# 在此过程中遇到的问题及解释

## 4.1 关于解压缩命令

**tar**

-c: 建立压缩档案  
-x：解压  
-t：查看内容  
-r：向压缩归档文件末尾追加文件  
-u：更新原压缩包中的文件

这五个是独立的命令，压缩解压都要用到其中一个，可以和别的命令连用但只能用其中一个。下面的参数是根据需要在压缩或解压档案时可选的。  
  
-z：有gzip属性的  
-j：有bz2属性的  
-Z：有compress属性的  
-v：显示所有过程  
-O：将文件解开到标准输出

下面的参数-f是必须的

-f: 使用档案名字，切记，这个参数是最后一个参数，后面只能接档案名。

# tar -cf all.tar \*.jpg  
这条命令是将所有.jpg的文件打成一个名为all.tar的包。-c是表示产生新的包，-f指定包的文件名。

# tar -rf all.tar \*.gif  
这条命令是将所有.gif的文件增加到all.tar的包里面去。-r是表示增加文件的意思。  
  
# tar -uf all.tar logo.gif  
这条命令是更新原来tar包all.tar中logo.gif文件，-u是表示更新文件的意思。  
  
# tar -tf all.tar  
这条命令是列出all.tar包中所有文件，-t是列出文件的意思  
  
# tar -xf all.tar  
这条命令是解出all.tar包中所有文件，-t是解开的意思

**压缩**

tar -cvf jpg.tar \*.jpg //将目录里所有jpg文件打包成tar.jpg

tar -czf jpg.tar.gz \*.jpg   //将目录里所有jpg文件打包成jpg.tar后，并且将其用gzip压缩，生成一个gzip压缩过的包，命名为jpg.tar.gz

 tar -cjf jpg.tar.bz2 \*.jpg //将目录里所有jpg文件打包成jpg.tar后，并且将其用bzip2压缩，生成一个bzip2压缩过的包，命名为jpg.tar.bz2

tar -cZf jpg.tar.Z \*.jpg   //将目录里所有jpg文件打包成jpg.tar后，并且将其用compress压缩，生成一个umcompress压缩过的包，命名为jpg.tar.Z

rar a jpg.rar \*.jpg //rar格式的压缩，需要先下载rar for linux

zip jpg.zip \*.jpg //zip格式的压缩，需要先下载zip for linux

**解压**

tar -xvf file.tar //解压 tar包

tar -xzvf file.tar.gz //解压tar.gz

tar -xjvf file.tar.bz2   //解压 tar.bz2

tar -xZvf file.tar.Z   //解压tar.Z

unrar e file.rar //解压rar

unzip file.zip //解压zip

**总结**

1、\*.tar 用 tar -xvf 解压

2、\*.gz 用 gzip -d或者gunzip 解压

3、\*.tar.gz和\*.tgz 用 tar -xzf 解压

4、\*.bz2 用 bzip2 -d或者用bunzip2 解压

5、\*.tar.bz2用tar -xjf 解压

6、\*.Z 用 uncompress 解压

7、\*.tar.Z 用tar -xZf 解压

8、\*.rar 用 unrar e解压

9、\*.zip 用 unzip 解压

参考：（<https://www.cnblogs.com/rickzhai/p/6432240.html>）

要注意压缩包是tar类型还是gz类型

## 4.2 关于环境变量

类似于Windows下的环境变量，本质含义就是让系统运行一个程序时，不仅仅要在当前目录下面寻找，还要到PATH指定的路径下寻找，此操作实际简化了程序执行的复杂度，提高了效率。

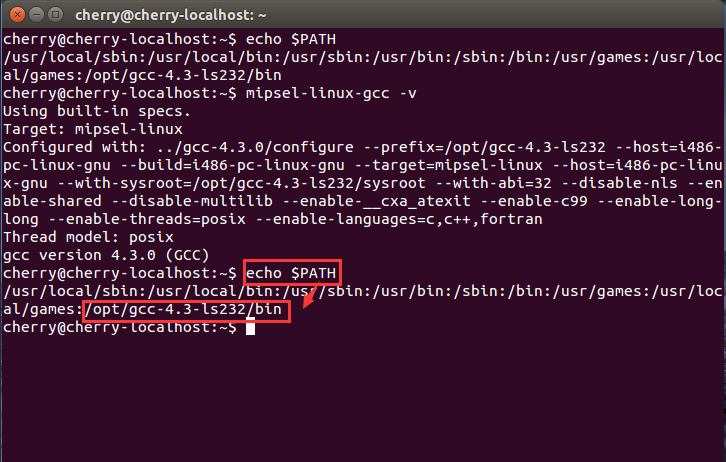
以上面的GCC路径为例，我们写入的PATH路径为：

export PATH=”$PATH:/opt/gcc-4.3-ls232/bin”

什么意思？可以把PATH看成一个字符串，用“/”、“：”、“$”等符号进行划分，系统根据划分的字符串寻找目标地址。

因为PATH是一个字符串，用“$PATH”表示引用这个字符串，就是表示PATH所指的内容；“：”是Linux中分隔符，相当于Windows中的“；”，因此该语句表示在原有的PATH路径下再加上/opt/gcc-4.3-ls232/bin路径，我们在终端中检验一下：

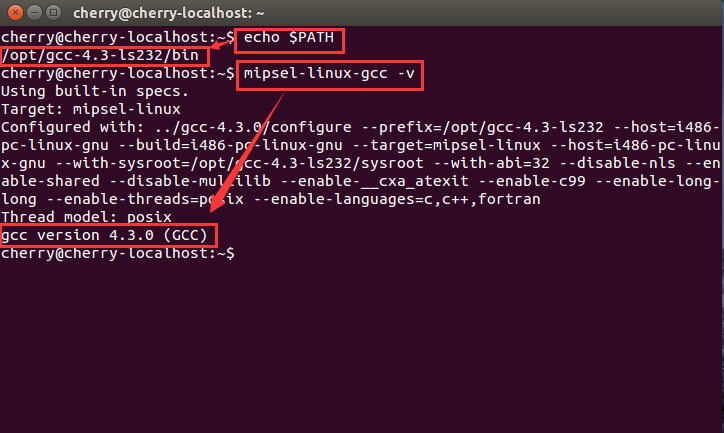
输入echo $PATH，结果如图：（echo是回显命令，是常见的脚本命令）



实际上，若要仅仅配置GCC，不需要加上之前的PATH路径，因此只要这样写即可：

export PATH=”/opt/gcc-4.3-ls232/bin”

我们在终端中再次输入echo $PATH，结果证明是正确的，gcc安装无误：



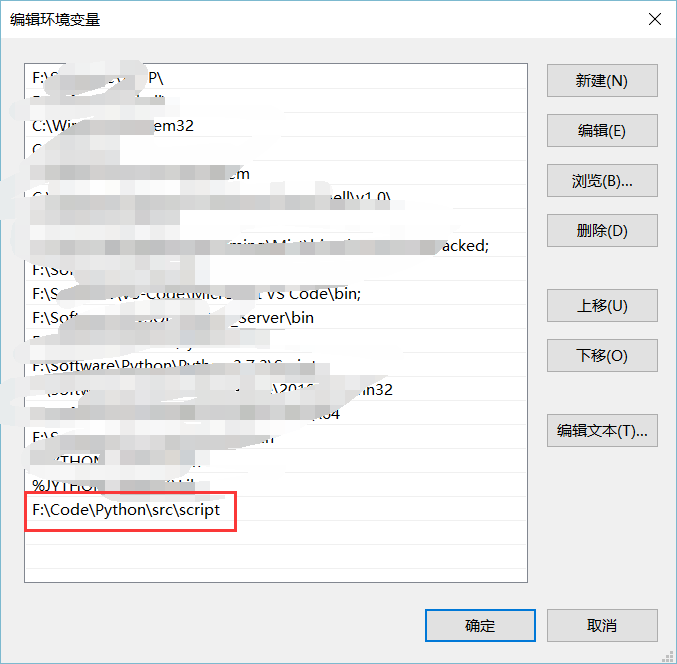
PATH环境变量的用处还有很多，比如有人喜欢通过命令行来执行程序，然后就不用进入文件夹执行了，其实利用PATH路径也是非常简单的，我在Windows环境下简单演示一遍：

若我想运行CSDN\_Visit.py（一个python脚本文件），在命令行里输入CSDN\_Visit.py

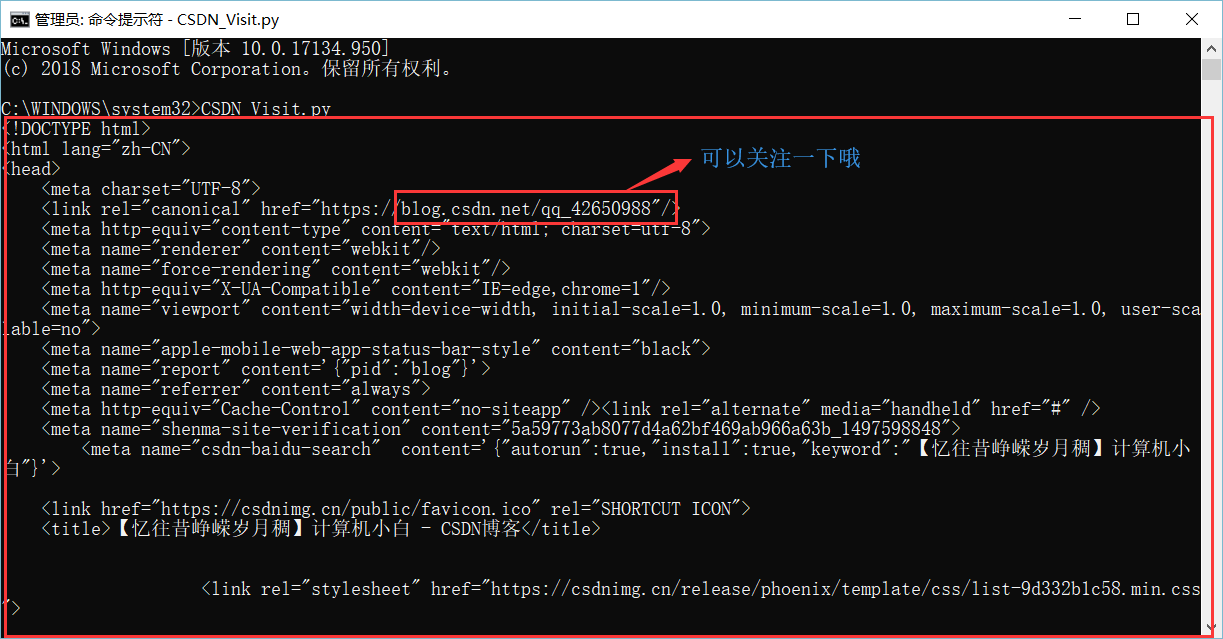


显然没有用，这时我把该文件放到F盘F:\Code\Python\src\script目录下：

在系统变量中加入F:\Code\Python\src\script



然后再次启动cmd，输入CSDN\_Visit.py 我们看到脚本已经成功执行



# 编译汇编指令

## 5.1 编译

我们先编写一组汇编指令，命名为inst\_rom.S



然后我们在终端中输入指令:

mipsel-linux-as -mips32 inst\_rom.S -o inst\_rom.o

表示用as工具将inst\_rom.S文件编译成inst\_rom.o文件

再输入：

mipsel-linux-ld -T ram.ld inst\_rom.o -o inst\_ram.om

表示将inst\_rom.o文件链接成inst\_rom.om文件

再输入：

mipsel-linux-objcopy -O binary inst\_rom.om inst\_ram.bin

根据.om文件得到了bin文件

再输入：

mipsel-linux-objdump -D inst\_rom.om > inst\_ram.asm

对汇编指令进行反汇编，得到与机器指令对应的二进制字

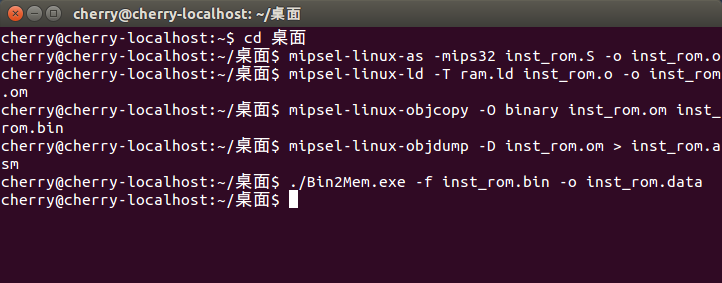
最后将Bin2MEM.exe拷贝到与inst\_rom系列文件相同的目录下，执行

./Bin2Mem.exe -f inst\_rom.bin -o inst\_rom.data

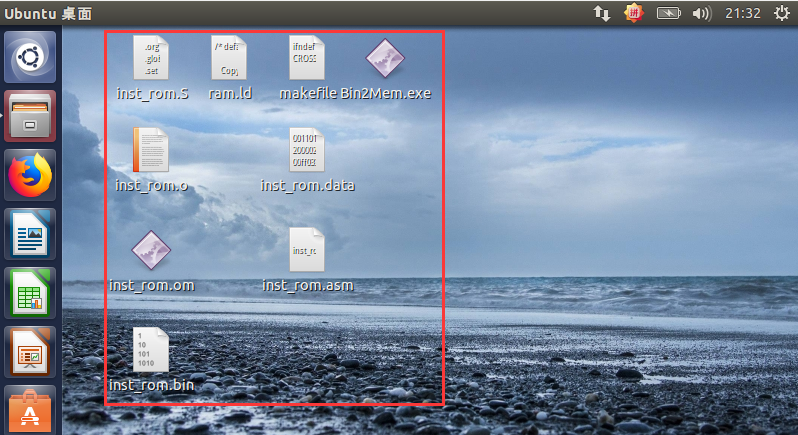
第一次执行应该需要修改Bin2Mem.exe文件的权限，输入：

Chmod 777 Bin2Mem.exe

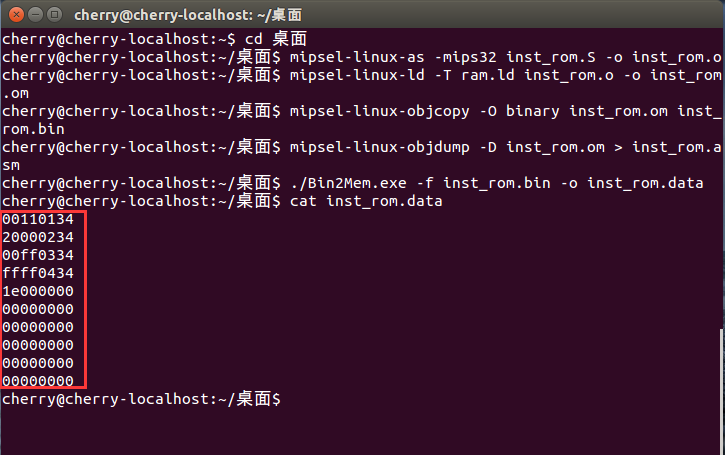
最终转化为与Vivado程序中读入的文件类型（.data），执行结果如图：



生成的文件：



我们打开inst\_rom.data文件查看：



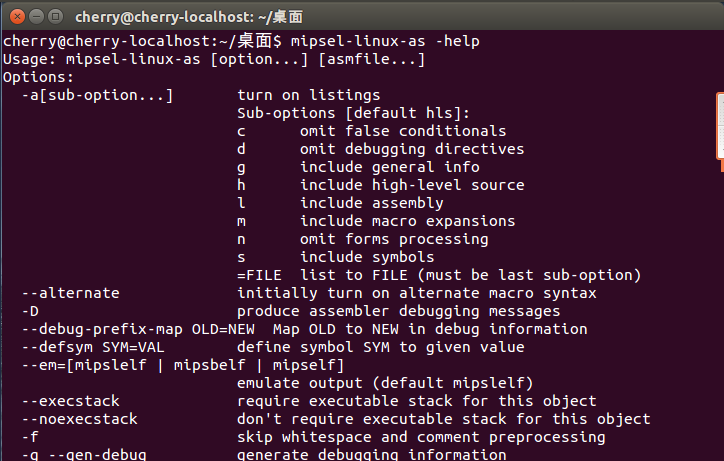
这便是我们一开始那几条汇编指令的机器指令。

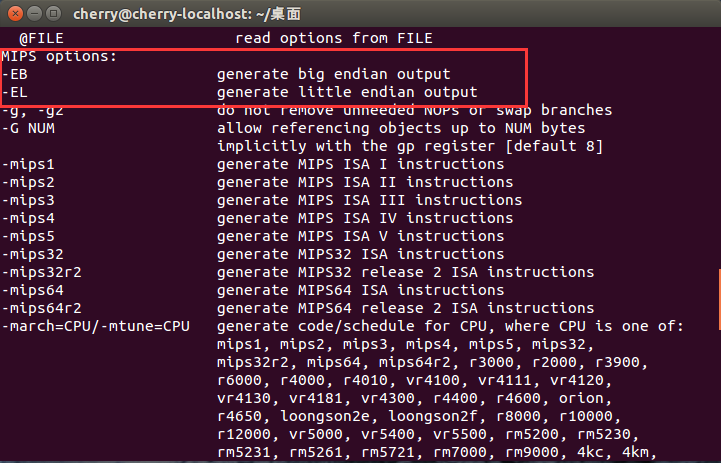
## 5.2 大小端地址

**大端（存储）模式**：是指一个数据的低位字节序的内容放在高地址处，高位字节序存的内容放在低地址处。

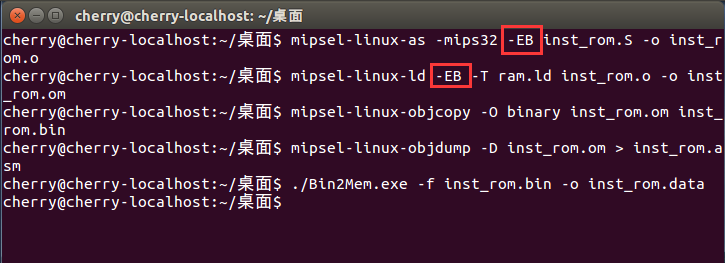
**小端（存储）模式**：是指一个数据的低位字节序内容存放在低地址处，高位字节序的内容存放在高地址处。（可以总结为“小小小”即低位、低地址、小端）

在上述的例子中，我们发现原来的机器指令34011100变成了00110134，高位数据放在了地址的高位，是小端存储，这样看上去不是很方便，我们可以修改编译指令改变最终编译的结果，输入mipsel-linux-as -help查看帮助

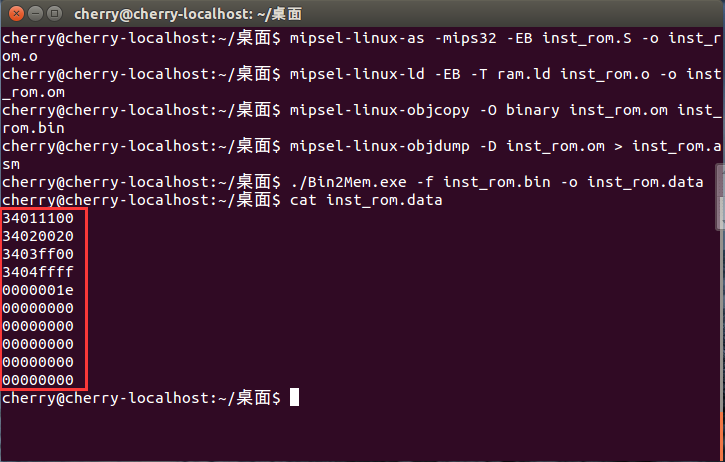




我们修改指令：



查看inst\_rom.data文件：



已经修改为大端模式了。

## 5.3 Make工具

每次都输入四个（加上.asm一共五个）指令显得很麻烦，我们可以编写一个脚本来自动执行命令。

**Make的工作原理**

当我们只输入make命令的工作流程是：

1. make会在当前目录下找名字叫“Makefile”或“makefile”的文件；

2. 如果找到，它会找文件中的第一个目标文件（target），在上面的例子中，他会找到“inst\_rom.data”这个文件，并把这个文件作为最终的目标文件；

3. 如果inst\_rom.data文件不存在，或是.data所依赖的后面的 .om 文件的文件修改时间要比.data这个文件新，那么make会执行下面定义的命令来生成.data文件；

4. 如果.data所依赖的.om文件也存在，那么make会在当前文件中找目标为.om文件的依赖性，如果找到再根据命令生成.om文件（这是一个递归的过程）；

如果在找寻的过程中，出现了被依赖的文件找不到的错误，那么make就会直接退出，并报错。

如果在一条依赖链中，比如：A依赖B，B依赖C，C依赖D。那么当D更新后，make发现D比C新则会重新构建C，以此类推，最终A也会被更新。

简单的说，makefile带来的好处就是——自动化编译，只要一个make命令，所有工程和文件自动编译，类似于Shell的.sh和cmd的.bat

我们编写如下代码：

ifndef CROSS\_COMPILE

CROSS\_COMPILE = mipsel-linux-

endif

CC = $(CROSS\_COMPILE)as

LD = $(CROSS\_COMPILE)ld

OBJCOPY = $(CROSS\_COMPILE)objcopy

OBJDUMP = $(CROSS\_COMPILE)objdump

OBJECTS = inst\_rom.o

export CROSS\_COMPILE

all:inst\_rom.data inst\_rom.om inst\_rom.o inst\_rom.bin inst\_rom.asm

%.o:%.S

$(CC) -mips32 -EB $< -o $@

inst\_rom.om:ram.ld $(OBJECTS)

$(LD) -EB -T ram.ld $(OBJECTS) -o $@

inst\_rom.bin:inst\_rom.om

$(OBJCOPY) -O binary $< $@

inst\_rom.asm:inst\_rom.om

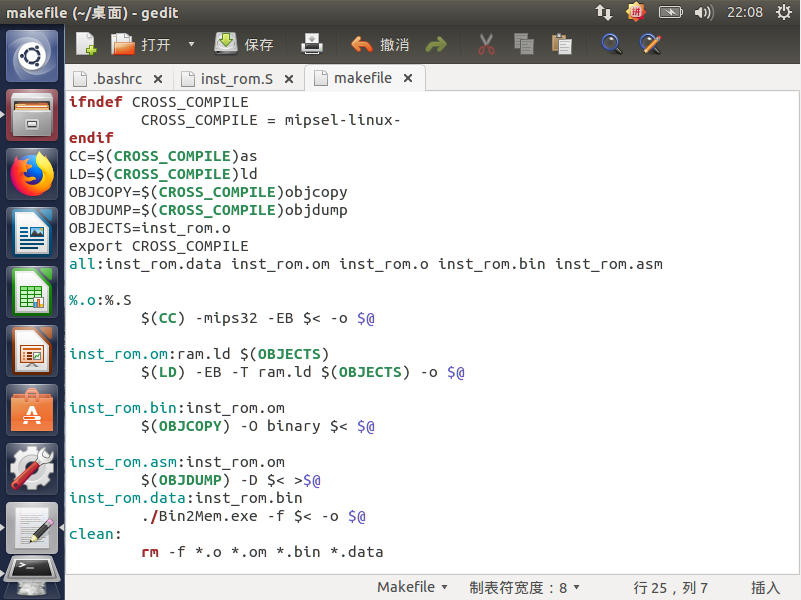
$(OBJDUMP) -D $< >$@

inst\_rom.data:inst\_rom.bin

./Bin2Mem.exe -f $< -o $@

clean:

rm -f \*.o \*.om \*.bin \*.data



其中 $< 表示第一个依赖文件的名称

$@ 表示目标的完整名称

至此，我们用gcc编译汇编指令生成机器码的过程就做完了。

# 附录

## 6.1 ram.ld代码

MEMORY

{

ram(RW) : ORIGIN = 0x00000000, LENGTH = 0x00001000

}

SECTIONS

{

/\*

For some reason the linker script can't see the \_reset\_vector symbol

(even if we declare it global), so we explicitly set it. \*/

.text :

{

\*(.text)

} > ram

.data :

{

\*(.data)

} > ram

.bss :

{

\*(.bss)

} > ram

.stack ALIGN(0x10) (NOLOAD):

{

\*(.stack)

\_ram\_end = .;

} > ram

}

ENTRY (\_start)

## 6.2 Bin2Mem.c代码

#include <stdlib.h>

#include <stdio.h>

char \*option\_invalid = NULL;

char \*option\_file\_in = NULL;

char \*option\_file\_out = NULL;

FILE \*file\_in\_descriptor = NULL;

FILE \*file\_out\_descriptor = NULL;

void exception\_handler(int code) {

switch (code) {

case 0:

break;

case 10001:

printf("Error (10001): No option recognized.\n");

printf("Please specify at least one valid option.\n");

break;

case 10002:

printf("Error (10002): Invalid option: %s\n", option\_invalid);

break;

case 10003:

printf("Error (10003): No input Binary file specified.\n");

break;

case 10004:

printf("Error (10004): Cannot open file: %s\n", option\_file\_in);

break;

case 10005:

printf("Error (10005): Cannot create file: %s\n", option\_file\_out);

break;

default:

break;

}

if (file\_in\_descriptor != NULL) {

fclose(file\_in\_descriptor);

}

if (file\_out\_descriptor != NULL) {

fclose(file\_out\_descriptor);

}

exit(0);

}

int main(int argc, char \*\*argv) {

int i=0,j=0;

unsigned char temp1,temp2,temp3,temp4;

unsigned int option\_flag = 0;

while (argc > 0) {

if (\*\*argv == '-') {

(\*argv) ++;

switch (\*\*argv) {

case 'f':

option\_flag |= 0x4;

argv ++;

option\_file\_in = \*argv;

argc --;

break;

case 'o':

option\_flag |= 0x8;

argv ++;

option\_file\_out = \*argv;

argc --;

break;

default:

option\_flag |= 0x1;

(\*argv) --;

option\_invalid = \*argv;

break;

}

}

argv ++;

argc --;

}

file\_in\_descriptor = fopen(option\_file\_in, "rb");

if (file\_in\_descriptor == NULL) {

exception\_handler(10004);

}

file\_out\_descriptor = fopen(option\_file\_out, "w");

if (file\_out\_descriptor == NULL) {

exception\_handler(10005);

}

while (!feof(file\_in\_descriptor)) {

fscanf(file\_in\_descriptor, "%c", &temp1);

fscanf(file\_in\_descriptor, "%c", &temp2);

fscanf(file\_in\_descriptor, "%c", &temp3);

fscanf(file\_in\_descriptor, "%c", &temp4);

if(!feof(file\_in\_descriptor)) {

fprintf(file\_out\_descriptor, "%02x", temp1);

fprintf(file\_out\_descriptor, "%02x", temp2);

fprintf(file\_out\_descriptor, "%02x", temp3);

fprintf(file\_out\_descriptor, "%02x", temp4);

fprintf(file\_out\_descriptor, "\n");

}

}

exception\_handler(0);

return 0;

}